PATENT

AMENDMENTS TO THE SPECIFICATION

Please replace the paragraph beginning on page 4, line 18 with the following amended paragraph:

C1

An embodiment of the present invention is illustrated by the block diagram of Figure 1, which shows a central processing unit (CPU) 100 in an exemplary 4-way superscalar processor of the present invention. A 4-way superscalar processor fetches, dispatches, executes and retires up to four instructions per processor cycle. As shown in Figure 1, central processing unit 100 includes two arithmetic logic units 101 and 102, a load/store unit 103, which includes a 9-deep load buffer 104 and an 8-deep store buffer 105, a floating point adder 106, a floating point multiplier 107, and a floating point divider 108. In this embodiment, a grouping logic circuit 109 dispatches up to four instructions per processor cycle. Completion unit 110 retires instructions upon completion. A register file (not shown), including numerous integer and float point registers, is provided with sufficient number of ports to prevent contention among functional units for access to this register file during operand fetch or result write-back. In this embodiment also, loads are non-blocking, i.e., CPU 100 continues to execute even though one or more dispatched laod instructions have not ~~complete~~ completed. When the data of the load instructions are returned from the main memory, these data can be placed in a pipeline for storage in a second-level cache. In this embodiment, floating point adder 106 and floating point multiplier 107 each have a 4-stage pipeline. Similarly, load/store unit 103 has a 2-stage pipeline. Floating point divider 108, which is not pipelined, requires more than one processor cycle per instruction.

Please replace the paragraph beginning on page 5, line 21 with the following amended paragraph:

C2

$$S(t) = \{ALU_1(t),\ ALU_2(t),\ LS(t),\ LB(t),\ SB(t),\ FA(t),\ FM(t),\ \text{~~FSD(t)~~}\ FDS(t)\}$$

Please replace the paragraph beginning on page 6, line 10 with the following amended paragraph:

C3

For pipelined functional units, such as floating point adder 106, the state is relatively more complex, consisting of the source and destination registers of the instructions in their ~~respectively~~ respective pipeline. Thus, for the pipelined units, i.e., load/store unit 103, load buffer 104, store buffer 105, floating point adder 106, and floating point multiplier 107, their respective states, at time t, LS(t), LB(t), SB(t), FA(t) and FM(t) can be represented by:

Please replace the paragraph beginning on page 6, line 24 with the following amended paragraph:

C4

Finally, floating point divider 108's state ~~FSD(t)~~ FDS(t)

Please replace the paragraph beginning on page 9, line 3 with the following amended paragraph:

C5

The state ~~FSD(t+1)~~ FDS(t+1) of floating point divider 108, in which the time required to execute an instruction can exceed [[an]] a processor cycle, is determined from state ~~FSD(t)~~ FDS(t) by:

~~*FSD(t+1)*~~ *FDS(t+1)* = ~~*FSD(t)*~~ *FDS(t)* *{if last stage}* *else null*

Please replace the paragraph beginning on page 9, line 14 with the following amended paragraph:

C6

In load buffer 104 and store buffer 105, since the pending read or write operation at the head of each queue need not complete within one processor cycle, the state LB(t+1) at time t+1 cannot be determined from the immediately previous state LB(t) at time t with certainty. However, since state LB(t+1) can only either remain the same, or reflect the movement of the pipeline by one stage, two possible approaches to determine state LB(t+1) can be used. First, a conservative approach would predict LB(t+1) to be the same as LB(t). Under this approach, when load buffer 104 is full, an instruction is not dispatched until the pipeline in load buffer 106 advances. An incorrect prediction, i.e. a load instruction completes during the processor cycle of time t, this conservative approach leads to a penalty of one processor cycle, since a load instruction could have been dispatched at time t+1. Alternatively, a more aggressive approach provides for both outcomes, i.e. load buffer 104 advances one stage, and load buffer 104 remains the

C6

same. Under this aggressive approach, grouping logic 109 is ready to dispatch a load instruction, such dispatch to be enabled by a control signal which indicates, at time t+1, whether a load instruction has in fact completed. This aggressive approach requires ~~more a~~ a more complex logic circuit more than the conservative approach.

Please replace the paragraph beginning on page 10, line 34 with the following amended paragraph:

C7

Once four candidate instructions for an instruction group are identified, intra-group data dependency checking can begin. Because of the constraint against instruction group merging described above, i.e., all instructions in an instruction group must be dispatched before an instruction from a subsequent instruction group can be dispatched, intra-group dependency checking can be accomplished in a pipelined fashion. That is, intra-group dependency checking can span more than one processor cycle and all inter-group dependency checking can occur independently of ~~inter-group~~ intra-group dependency checking. For the purpose of intra-group dependency check, each instruction group can be represented by: